Towards a Scalable Architecture for Real-Time Volume Rendering

Towards a Scalable Architecture for Real-Time Volume Rendering

Hanspeter Pfister, Arie Kaufman, and Frank Wessels
State University of New York at Stony Brook
U.S.A. *

Abstract

In this paper we present our research efforts towards a scalable volume rendering architecture for the real-time visualization of dynamically changing high-resolution datasets. Using a linearly skewed memory interleaving we were able to develop a parallel dataflow model that leads to local, fixed-bandwidth interconnections between processing elements. This parallel dataflow model differs from previous work in that it requires no global communication of data except at the pixel level. Using this dataflow model we are developing Cube-4, an architecture that is scalable to very high performances and allows for modular and extensible hardware implementations.

1 Introduction

Volume visualization has become a key technology in the interpretation of the large amounts of volumetric data generated by acquisition devices such as biomedical scanners, by supercomputer simulations, or by synthesizing (voxelizing) geometrical models using volume graphics techniques [9, 11]. It encompasses an array of techniques for extracting meaningful information from the datasets and displaying it in a visual form. Of particular importance for the manipulation and display of static and dynamic volumetric objects are the interactive change of projection and rendering parameters, real-time display rates, and in many cases the possibility to view changes of a dynamic dataset over time, a process that is often called 4D (spatial-temporal) visualization.

Users in modern scientific, industrial, and medical environments often have direct access to acquisition devices for volumetric data, including CT, MRI, and ultrasound scanners and confocal microscopes. This warrants the development of stand-alone visualization systems that directly interface to these modalities. These integrated acquisition-visualization systems will allow their users to navigate around their 3D static data in real-time, or to view their temporally changing 4D data in real-time. Examples are the real-time visualization of a moving fetus or a beating heart under an ultrasound probe, real-time analysis of an in-vivo specimen under a confocal microscope, or the real-time study of in-situ fluid flow or crack formation in rocks under Computed Microtomograph (CMT), which is under development by a DOE project and estimated to deliver $400 \times 256 \times 256$ samples at 15Hz.

The main goal of our research is to develop a special-purpose real-time volume visualization architecture for high-resolution datasets that will support 4D volume visualization. We have set the following design objectives based on what we believe to be important features of a real-time volume rendering system:

Real-Time Frame Rates: To create the illusion of smooth motion, the image must be updated a minimum of 24 times per second. The architectures presented in this paper aim at achieving projection rates of 30 frames per second.

4D Visualization: The architecture has to allow for the real-time input of volumetric data without precomputations. The overall latency of the system should be no more than one frame time.

High-Resolution Datasets: The architecture has to be able to visualize dataset resolutions of 512^3 voxels or higher in real-time.

Scalability: The design should be modular, and the performance should ideally scale almost linearly in the number of modules.

High Image Quality: The images must be of high quality, including surface shading, depth cues, and

*Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794–4400, U.S.A, email: pfister,ari,wessels@cs.sunysb.edu

the provision of transparency. Special care has to be taken to avoid image artifacts such as spatial or temporal aliasing.

Flexibility: The algorithm and hardware should be flexible enough to allow for the interactive change of parameters such as shading, data segmentation, and projection modes.

As we will explain in Section 2, current general-purpose systems fall short of achieving these goals. Our research may yield two important contributions towards real-time visualization systems for volume data. On the one hand, we are conducting research towards the design of add-on volume rendering accelerators for general-purpose machines. The same way as the special requirements of traditional computer graphics led to the proliferation of special-purpose graphics engines, primarily for accelerating polygon rendering, volume visualization lends itself to the development of special-purpose volume rendering engines. On the other hand, we are developing special-purpose volume rendering hardware that can be embedded into modern acquisition devices. This work may lead to the direct integration of volume visualization hardware with acquisition devices, much in the same way as fast signal processing hardware became part of today's scanning devices.

2 Related Work

One way to try to meet the above design objectives is to employ large-scale parallelism on general-purpose supercomputers. Advantages of this approach are the flexible programming environment and the ability to integrate the simulation and the visualization on the same machine. However, the state-of-the-art in parallel volume rendering is in the range of one to (at most) 10 frames per second of low-resolution datasets [20, 16, 24, 26]. For interactivity, the image generation latency (i.e., the time between the request and the receipt of the completed image) is more important than the image generation frame rate. Since most volume rendering algorithms require very little repeated computation per voxel, data movement and interprocessor communication account for a significant portion of the overall performance overhead. This greatly impacts the latency, making current supercomputers inappropriate for interactive use. Furthermore, supercomputers seldom contain frame buffers and due to their high cost are typically shared by many users. Each user is assigned only a partition of the machine, thereby further inhibiting fast volume rendering rates. Most users have access through network connections without interactive data input and output rates.

A few researchers have implemented volume rendering algorithms on experimental special-purpose high-performance graphics systems. A 1024-processor Princeton Engine [3], a real-time video system simulator, has achieved 30 frames per second for rotations around the z-axis of 128^3 datasets [23]. The Pixel-Planes 5 multiprocessor graphics system is capable of rendering $56 \times 128 \times 128$ datasets at 20 frames per second [19, 29, 25]. These machines were specially designed for real-time video simulations or high-quality rendering of large polygonal scenes, respectively. They are highly-parallel machines, where most of the hardware resources are spent on video processing, polygon rasterization, and z-buffer image composition [3, 4, 18]. This high degree of specialization makes them unsuited for direct volume rendering applications. High-resolution datasets are unable to fit into the physical memory of the machines, and their cost and size prevent integration into desktop or deskside systems.

A recently developed method on a deskside system uses the texture-memory of a high-end four Raster Manager RalityEngine Onyx with a 150 MHz R4400 to render unshaded images from a $64 \times 512 \times 512$ dataset with 8-bit voxels in 0.1 sec [2]. This approach suffers from several limitations. The texture hardware does not support gradient estimation, and high-resolution datasets or datasets with more than 8 bits per voxel do not fit into the texture memory. The limited texture buffer bandwidth inhibits real-time input, and the required texture hardware is large and expensive.

Several researchers have proposed special-purpose volume rendering architectures [9, Chapter 6]. VOGUE and VIRIM are more recent ray-casting architectures. VOGUE [13], a modular add-on accelerator, is estimated to achieve 2.5 frames per second for 256^3 datasets. VOGUE will require 64 boards and a 5.2 GB/sec ring-connected cubic network to achieve 20 frames per second of 512^3 datasets. VIRIM [5], a programmable ray-casting engine, requires duplication of data and 16 boards for rendering $128 \times 256 \times 256$ datasets at 10 frames per second.

In this paper we present our first steps towards Cube-4, a scalable volume rendering architecture that meets all of our design objectives. It will provide users with real-time viewing from arbitrary parallel and perspective directions, control of rendering and projection parameters, and mechanisms for visualizing internal and surface structures. Cube-4 is based on a data-parallel algorithm for ray-casting of a volume buffer of voxels which is stored as a skewed distributed memory. The architecture performs interpolation of sampled points along rays, shading, and compositing of the sampled points to generate the pixel values. We believe that this approach will lead to the development of the first scalable volume visualization architecture that will support real-time, high-quality, volume rendering of high-resolution (for example 1024^3) volume data.

3 Ray-Casting in Real-Time

Ray-casting is the volume visualization algorithm underlying our research efforts. It is the most commonly used volume rendering technique. It simulates optical projections of light rays through the dataset (see [17] for a description of the underlying optical models). In a typical algorithm rays are cast from the viewpoint through each pixel of the view-plane into the volume data. At sample locations along each ray the data is usually tri-linearly interpolated using values of eight surrounding voxels. Central differences of voxels around the sample point yield a gradient as a surface normal approximation. Using the gradient and the interpolated sample value, a local shading model is applied and a sample opacity is assigned. This opacity classification allows for interactive data segmentation without any pre-computations. Samples along the ray are composited into pixel color values to produce an image [15].

However, the high computational cost of ray-casting makes it difficult for sequential implementations on general-purpose computers to deliver the targeted level of performance. This situation is aggravated by the continuing trend towards higher and higher resolution datasets. For example, to render a high-resolution dataset of 1024^3 16-bit voxels at 30 Hz requires 2 GBytes of storage, a memory transfer rate of 60 GBytes per second, and approximately 300 billion instructions per second, assuming 10 instructions per voxel per projection.

The fastest single-chip processors currently available compute approximately 300 million floating-point or integer operations per second, and the fastest DRAM memory systems have cycle times of approximately 70ns. The performance requirements for this modest example, therefore, exceed by far the capabilities of a single processor or single memory system. Consequently, it is imperative to use parallelism, both in the form of pipelining and unit replication, for a system that tries to achieve real-time performance. The four most compute-intensive parts of ray-casting are dataset traversal, interpolation, gradient estimation and shading, and compositing. A high-performance ray-casting engine must perform all of them in parallel.

To access the data in parallel requires a distributed memory system. Cube-1, a first generation hardware prototype, is based on a specially organized cubic frame buffer (CFB) [10], which has also been used in all subsequent generations of the Cube architecture developed at SUNY Stony Brook. It uses a simple linear memory skewing, where a voxel with space coordinates (x, y, z) is being mapped onto the k-th memory module by:

$$k = (x + y + z) \bmod n \qquad 0 \leq k, x, y, z \leq n - 1.$$

This 3D skewed organization of the n^3 voxel CFB enables conflict-free access to any beam (i.e., a ray parallel to a main axis) of n voxels. A fully operational printed circuit board (PCB) implementation of Cube-1 is capable of generating orthographic projections of 16^3 datasets from a finite number of predetermined directions in real-time. Cube-2 is a single-chip VLSI implementation of this prototype. An extension of the orthographic projection mechanism enables arbitrary parallel projections at a predicted performance of 16 frames per second for 512^3 datasets [1].

One important problem that inhibits real-time ray-casting is the very frequent and mostly random accesses to the volume memory. The same voxel has to be fetched several times for each projection. The reasons for these multiple accesses are twofold. First, it is the non-uniform mapping of sample point onto voxels. Due to either a small sampling step along a ray or a high pixel density, multiple samples along the same ray or of neighboring rays may map onto the same voxel. Second, it is the overlap of voxel neighborhoods for tri-linear interpolation and gradient estimation calculations, that is, the same voxel may be involved in multiple calculations. This leads to multiple and redundant data accesses to the volume memory. In message passing computation models it also leads to excessive interprocessor communication.

In our previous work we have studied and developed a template-based (lookup-table based) ray-casting approach for which there is a one-to-one mapping of sample locations onto voxels [28]. 26-connected discrete rays are pre-generated from continuous rays using a 3D variation of Bresenham's algorithm modified for non-integer endpoints [9]. This algorithm guarantees constant stepping with a unit increment along the major viewing direction. The stepping in the two non-major directions is stored in lookup tables, so-called x- and y-templates. This approach allows for efficient projections onto the base-plane, which is the face of the volume memory that is most perpendicular to the viewing direction. The resulting distorted image on the base-plane is then 2D warped onto the image plane. Schröder and Stoll [23] and Lacroute and Levoy [14] have used similar approaches on massively parallel machines and graphic workstations, respectively. These implementations achieve interactive performances for low-resolution datasets. However, they use a preprocessing step and data duplication to calculate the gradient field or to generate color and opacity volumes and are thus unsuitable for high-resolution 4D visualization.

Consequently, we extended our template-based approach to include new methods for tri-linear interpolation and gradient estimation in order to access every voxel exactly once per projection. In Cube-3, we introduced a way to perform tri-linear interpolation using the template-generated discrete voxel rays. Because of

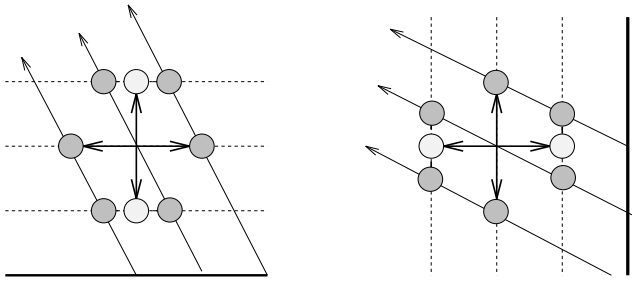
the discrete steps along the rays, the voxel neighborhood around each sample location may be non-cubic or sheared. We avoid fetching any additional voxels from the volume memory using sheared tri-linear interpolation [22]. Instead of specifying the sample location with respect to a corner voxel of the interpolation neighborhood, we factor the tri-linear interpolation into four linear and one bi-linear interpolation using the possibly sheared voxel neighborhood between rays. The interpolation weights can be pre-computed and stored in the x- and y-templates.

We conducted several experiments with the sheared tri-linear interpolation method using a CT study of a cadaver head of size $256 \times 256 \times 225$ voxels at 8-bit per voxel (dataset provided courtesy of North Carolina Memorial Hospital). As error measure for the comparison between the final images of traditional tri-linear interpolation and sheared tri-linear we use the average Euclidean distance of RGB values between corresponding pixels. During a full rotation of the dataset the average error in percentage stays below 0.3 %. (See [22] for more detailed results).

We also developed new ways of gradient estimation using interpolated samples from neighboring rays above, below, and along the current ray. This so-called ABC gradient estimation avoids any additional fetching of voxels from the volumetric dataset [22]. Traditional gradient estimation techniques compute a gray-level gradient by taking local differences between voxel values in all three dimensions at the original grid points [7]. Tri-linear interpolation is used to obtain a gradient at the sample location. In order to avoid these additional memory accesses to the dataset, we use central differences between the tri-linearly interpolated sample values on rays on the immediate left, right, above and below, as well as the values along the current ray.

After we investigated several ABC gradient estimation methods using 6, 10, or 26 samples of neighboring rays [22], we developed the following so called 12-neighborhood gradient method. It allows to calculate highly accurate gradients that are parallel to the primary axes of the volume memory. Figure 1 shows the basic idea using a two-dimensional drawing. The lightly shaded samples are interpolated using linear interpolation of voxels from the current ray and the left and right ray, respectively. This is indicated by dashed lines in Figure 1. Whereas in 2D only two linear interpolations are performed, we need two bi-linear interpolations in 3D, involving a total of 12 samples in the calculations.

For an analysis of the error due to different ABC gradient estimation techniques we used a dataset of a voxelized sphere. The sphere was scan-converted using the volume sampling method described in [27]. The surface intersection points during ray-casting are obtained by thresholding, i.e., as soon as a certain sample value is



(a) 12-neighborhood, Horizontal Base-Plane **(b) 12-neighborhood, Vertical Base-Plane**

Figure 1: 12-neighborhood Gradient.

exceeded we calculate the gradient at that point. Each gradient is compared to the true geometric surface normal. As error measure we use the magnitude of angular difference between the two vectors averaged over all surface intersection points.

Figure 2 shows the results of rotating the sphere around a vertical axis between 0^0 and 90^0 in steps of 5^0. During the rotation of the sphere the average error

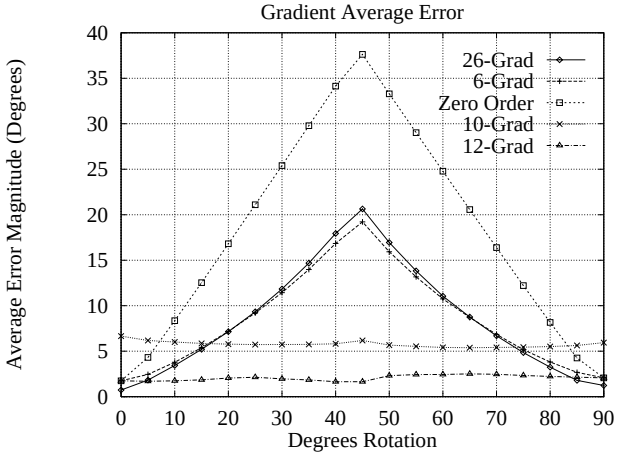


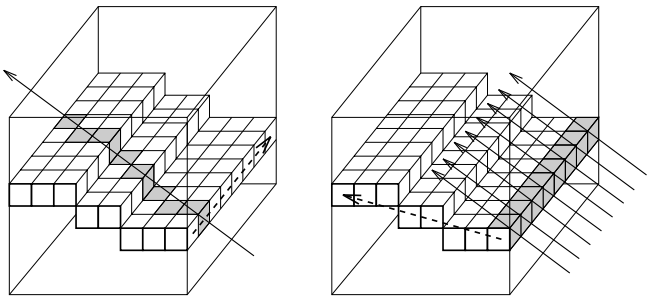
Figure 2: Average error magnitude of comparing different ABC gradient estimation schemes to the true analytic normal on a voxelized sphere.

for 12-neighborhood gradient estimation stayed below 3^0 when compared to the analytic normal. This error is substantially lower than for other ABC gradient estimation techniques or for zero-order central difference gradients.

Both the sheared tri-linear interpolation and the ABC gradient estimation method do not require any pre-computation, reduce the number of accesses to the volume memory to one per voxel per projection, and allow for efficient hardware implementations [21, 6].

4 Parallel Ray-Casting

In order to attain the memory bandwidth required for real-time ray-casting, we developed data parallel versions of the real-time algorithms presented so far. The parallelism exploited by these algorithms is best described as ray-parallel and beam-parallel. Figure 3 compares the two approaches. In the ray-parallel approach,



a) Ray-Parallel **b) Beam-Parallel**

Figure 3: Two different approaches to parallel ray-casting. Shaded voxels are processed simultaneously. The dashed arrows indicate the direction the algorithm proceeds in subsequent timesteps.

shown in Figure 3a, voxels and samples of a single ray are processed simultaneously. Using a pipelined implementation a base-plane pixel is completed every iteration. The Cube-3 architecture [21, 22] is a highly-pipelined implementation of this ray-parallel approach. Figure 4a gives an overview of the conceptual architecture of Cube-3. It uses the same linear skewing of the CFB as in Cube-1 and Cube-2. A high-speed global communication network, the Fast Bus, aligns and distributes voxels from the CFB to tri-linear interpolation units (TRILIN). Using coherency among neighboring rays, special shading units (Shaders) estimate the gradient at each sample location and assign color and opacity to the samples. A circular cross-linked binary tree of voxel combination units (VCUs) composites all samples into the final pixel color. Estimated performance for arbitrary parallel and perspective projections is 30 frames per second for 512^3 datasets.

The global communication network in Cube-3, however, limits its scalability. For each projection all dataset voxels have to be transmitted over the Fast Bus. The required bus bandwidth is high and increases with $O(n^3)$, n being the dataset resolution. The interconnection of VCUs in a wrap-around binary tree fashion leads to problems at chip and board boundaries for higher resolution datasets. It has been our goal to address these issues, mainly to simplify the datapath and control logic, decrease the machine size, and enhance the scalability.

We developed a new data-parallel approach to ray-casting shown in Figure 3b. Instead of processing individual rays it simultaneously manipulates a group of rays. We call this approach beam-parallel, because the beams intersected by the viewing rays of a base-plane scanline are fetched consecutively in the direction of the major viewing axis. The n pixels of a base-plane scanline are completed after n steps, after which the following scanline is processed.

The data is stored in a linearly skewed cubic frame buffer that allows for conflict-free parallel access to any beam of voxels. All rays corresponding to a scanline on the base-plane reside inside a so called projection ray plane (PRP). To generate the scanline pixels corresponding to each PRP, we first fetch all voxels of a PRP using the conflict-free beam access mechanism. The next step is to compute beams of interpolated continuous ray samples using the voxels of four beams coming from two adjacent PRPs. In order to be able to fetch beams from both PRPs during the same timestep we need to buffer one of them, which can trivially be accomplished by using a first-in-first-out (FIFO) buffer connected to the memory modules. After two timesteps a total of four beams has been fetched from the memory and FIFOs, and a beam of interpolated samples can be computed.

The interpolated beams are forwarded to the gradient estimation units, where they are stored inside ABC buffers. These ABC buffers can easily be implemented using a similar FIFO buffering as for the PRPs. Since the above samples come directly from the tri-linear interpolation units we do not need to store them, but we need only two FIFO buffers for the current and below planes. To estimate the gradient around a given sample location of the current buffer we use samples along the direction of the ray from the current buffer, samples inside the current plane perpendicular to the direction of the ray from neighboring gradient estimation units, and samples from the above and below buffers. These samples are used for a 12-neighborhood grey-level gradient estimation. Using a pipelined implementation of the 12-neighborhood gradient and n ABC gradient units in parallel we can estimate n gradients around the n sample locations of the current buffer at every timestep.

After the gradient estimation follows the shading of the samples. With the gradient and the light vector description we produce n shaded samples of n continuous rays per timestep. This assumes that we can perfectly pipeline the shading calculations, which may be non-trivial for higher order, e.g., Phong, shading models. However, our simulations show very satisfactory results using a simple diffuse shading model. Other researchers have proposed fully pipelined Phong shading architectures [12].

In order to generate the final base-plane pixel values corresponding to the current PRP, we perform alpha blending or compositing using an opacity lookup table

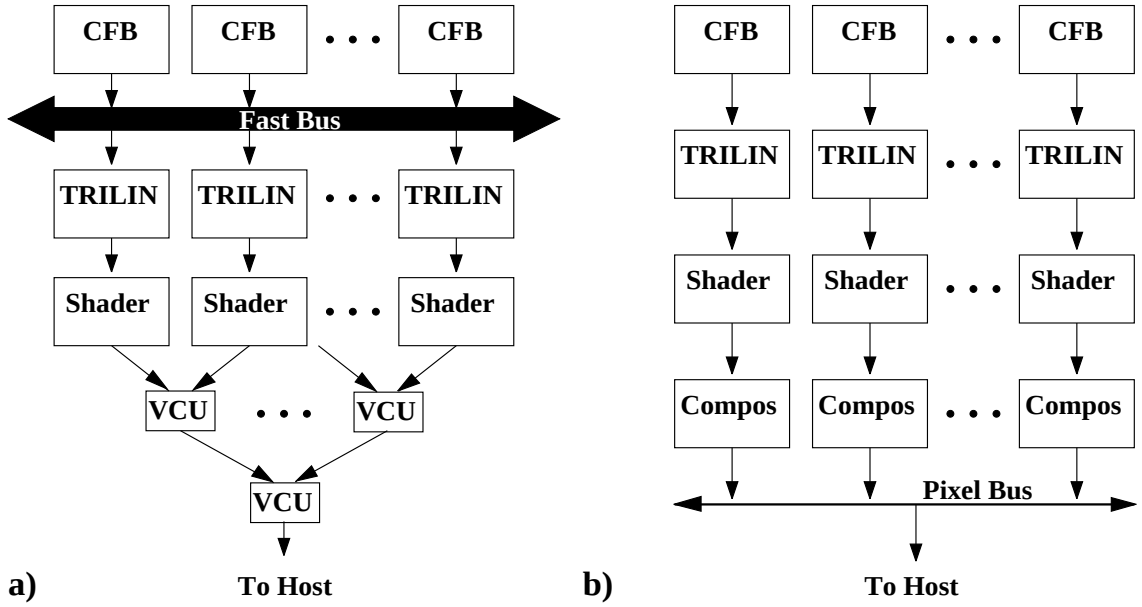


Figure 4: The Cube-3 (a) and Cube-4 (b) conceptual architectures.

(transfer function) and the shaded sample value. We use simple accumulating adders to perform the compositing, yielding n final base-plane pixel values every n timesteps. Operations like first/last opaque, maximum or average projection can also be implemented in the compositors. As a last step we have to transmit the base-plane pixels to the host where the transformation and resampling onto the view-plane is performed. Since in our example we get n base-plane pixels every n timesteps we can easily transmit one pixel per timestep to the host where they are buffered before the final 2D warp.

Our next generation architecture, called Cube-4, is a hardware implementation of this beam-parallel dataflow model. In order to allow conflict-free access to any beam, Cube-4 uses the same skewed memory organization as the previous Cube architectures. Due to this skewing the data is not directly forwarded to the nearest neighbor processing element, but with localized, fixed-bandwidth connections between memory, tri-linear interpolation, shading and compositing units. This is conceptually illustrated in Figure 4b. Instead of processing individual rays, we manipulate groups of rays in a beam-parallel fashion. As a result, the rendering pipeline is directly connected to the memory. Accumulating compositors replace the binary compositing tree. A pixel-bus collects and aligns the pixel output from the compositors.

Because only $O(n^2)$ pixels per projection are being globally transferred, the Cube-4 architecture is scalable to high dataset resolutions. Instead of global voxel communication over a Fast Bus it uses a simple, easy to implement pixel bus with only moderate bandwidth requirements. The lack of global communication between n parallel units at the same timestep allows for inexpensive implementations and high packing densities. The estimated performance and scalability of Cube-4 will be discussed in further detail in the following section.

5 Performance Estimation

The fixed datapath connections and the simple control make it easy to exploit parallelism and pipelining at every stage of Cube-4. The performance is thereby solely limited by the speed of the memory. Commercial DRAMs typically have a random access frequency of $f_d = 8.33$ MHz (assuming 120 ns cycle time). This allows for 30 projections per second with dataset dimensions $n \leq 512$, using n off-the-shelf standard DRAMs. High-resolution implementations require the higher memory access speeds delivered by synchronous DRAMs (SDRAMs) or enhanced DRAMs (EDRAMs) [8]. Currently, these high-speed memory devices achieve an average access frequency of $f_d = 33$ MHz, allowing for 1024^3 implementations.

By clocking the DRAMs at their maximum frequency f_d it is possible to reduce the number of physical memory chips, each of them storing more of the data. As an example, a 128^3 machine using standard DRAMs with

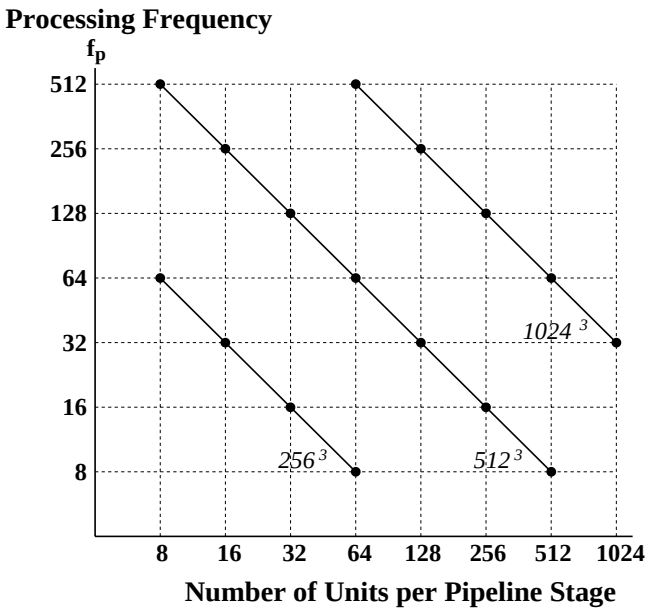


Figure 5: Scalability curves for 30 projections per second.

$f_d = 8.33$ MHz requires only 8 physical memory chips instead of 128 for the fully parallel version. Each of the chips stores 16 times the data of the fully parallel version.

In order to further reduce the hardware complexity, we can increase the processing frequency and use fewer processing units that operate on multiple data items in a time-sliced fashion. Figure 5 shows the tradeoff between the number of processing units per pipeline stage, $\frac{n}{m}$, and the processing frequency, f_p, for three dataset resolutions. Depending on available technology it is possible to combine two or more stages of the pipeline into one physical unit. It can be seen from Figure 5 that 32 units per pipeline stage running at 128 MHz suffice to implement a 512^3 machine. Using fast DRAMs such a design could be implemented using 128 memory chips, bringing it into the realm of VME board sizes. Similarly, we can achieve EISA board implementations for 256^3 datasets using 8 units per pipeline stage running at 64 MHz and 16 fast DRAMs. Note that we assume true real-time projection rates of 30 frames per second.

6 Conclusions and Future Work

We have presented our design objectives and first steps towards a special-purpose scalable architecture that can deliver real-time high-quality ray casting of high-resolution datasets. Using a novel data-parallel volume rendering algorithm called beam-parallel ray-casting we are able to avoid any global communication of voxels, and only require a pixel-bus of moderate bandwidth. The resulting Cube-4 architecture is scalable, modular in design, and has the potential of rendering high-resolution datasets, such as 1024^3 16-bit voxels, at 30 frames per second. Using sheared tri-linear interpolation and 12-neighborhood ABC gradient estimation avoids any pre-computations and allows for 4D visualization of dynamically changing data.

We will continue the development of efficient algorithms and scalable real-time architectures for volume rendering. Our future research has two main components. First, the further development and analysis of beam-parallel ray-casting algorithms and the resulting Cube-4 architecture. Second, the implementation of a reduced resolution Cube-4 prototype to test our algorithms and architectural studies.

References

[1] Bakalash, R., Kaufman, A., Pacheco, R., and Pfister, H. An extended volume visualization system for arbitrary parallel projection. In *Proceedings of the 1992 Eurographics Workshop on Graphics Hardware* (Cambridge, UK, Sept. 1992).

[2] Cabral, B., Cam, N., and Foran, J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In In *1994 Workshop on Volume Visualization* (Washington, DC, Oct. 1994), pp. 91–98.

[3] Chin, D., Passe, J., Bernard, F., Taylor, H., and Knight, S. The princeton engine: A real-time video system simulator. *IEEE Transactions on Consumer Electronics 34*, 2 (1988), 285–297.

[4] Fuchs, H., Poulton, J., Eyles, J., Greer, T., Goldfeather, J., Ellsworth, D., Molnar, S., Turk, G., Tebbs, B., and Israel, L. Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics 23, No. 3* (July 1989), 79–88.

[5] Günther, T., Poliwoda, C., Reinhard, C., Hesser, J., Männer, R., Meinzer, H.-P., and Baur, H.-J. VIRIM: A massively parallel processor for real-time volume visualization in medicine. In *Proceedings of the 9th Eurographics Hardware Workshop* (Oslo, Norway, Sept. 1994), pp. 103–108.

[6] Hesser, J., Männer, R., Knittel, G., Strasser, W., Pfister, H., and Kaufman, A. Three architectures for volume rendering. In To appear in *Proceedings of Eurographics '95* (Maastricht, The Netherlands, Sept. 1995), European Computer Graphics Association.

[7] HÖHNE, K. H., AND BERNSTEIN, R. Shading 3D-images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging MI-5*, 1 (Mar. 1986), 45–47.

[8] Special report on high-speed DRAMs. *IEEE Spectrum 29*, 10 (Oct. 1992), 34–57.

[9] KAUFMAN, A. *Volume Visualization*. IEEE CS Press Tutorial, Los Alamitos, CA, 1991.

[10] KAUFMAN, A., AND BAKALASH, R. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics & Applications 8*, 6 (Nov. 1988), 10–23. Also in Japanese, *Nikkei Computer Graphics*, 3, No. 30, March 1989, pp. 148–160.

[11] KAUFMAN, A., COHEN, D., AND YAGEL, R. Volume graphics. *IEEE Computer 26*, 7 (July 1993), 51–64.

[12] KNITTEL, G. VERVE: Voxel engine for real-time visualization and examination. In In *Computer Graphics Forum* (Sept. 1993), vol. 12, No. 3, pp. 37–48.

[13] KNITTEL, G., AND STRASSER, W. A compact volume rendering accelerator. In In *1994 Workshop on Volume Visualization* (Washington, DC, Oct. 1994), pp. 67–74.

[14] LACROUTE, P., AND LEVOY, M. Fast volume rendering using a shear-warp factorization of the viewing transform. *Computer Graphics, Proceedings of SIGGRAPH '94* (July 1994), 451–457.

[15] LEVOY, M. Display of surfaces from volume data. *IEEE Computer Graphics & Applications 8*, 5 (May 1988), 29–37.

[16] MA, K., PAINTER, J., HANSEN, C., AND KROGH, M. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics & Applications 14*, 4 (1994), 59–68.

[17] MAX, N. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June 1995), 99–108.

[18] MOLNAR, S., EYLES, J., AND POULTON, J. Pixelflow: High-speed rendering using image composition. *Computer Graphics 26*, 2 (July 1992), 231–240.

[19] NEUMANN, U. Interactive volume rendering on a multicomputer. In *1992 Symposium on Interactive 3D Graphics* (Cambridge, MA, Mar. 1992), ACM Computer Graphics, pp. 87–93.

[20] NIEH, J., AND LEVOY, M. Volume rendering on scalable shared-memory MIMD architectures. *Workshop on Volume Visualization* (Oct. 1992), 17–24.

[21] PFISTER, H., KAUFMAN, A., AND CHIUEH, T. Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization. In *Volume Visualization Symp. Proc.* (Wash., DC, Oct. 1994), pp. 75–83.

[22] PFISTER, H., WESSELS, F., AND KAUFMAN, A. Sheared interpolation and gradient estimation for real-time volume rendering. In *Proceedings of the 9th Eurographics Hardware Workshop* (Oslo, Norway, Sept. 1994), pp. 70–79.

[23] SCHRÖDER, P., AND STOLL, G. Data parallel volume rendering as line drawing. In In *1992 Workshop on Volume Visualization* (Boston, MA, Oct. 1992), pp. 25–31.

[24] SINGH, J. P., GUPTA, A., AND LEVOY, M. Parallel visualization algorithms: Performance and architectural implications. *IEEE Computer 27*, 7 (1994), 45–55.

[25] STATE, A., MCALLISTER, J., NEUMANN, U., CHEN, H., CULLIP, T., CHEN, D. T., AND FUCHS, H. Interactive volume visualization on a heterogeneous message-passing multicomputer. In *1995 Symposium on Interactive 3D Graphics* (Monterey, CA, Apr. 1995), pp. 69–74.

[26] VÉZINA, G., FLETCHER, P., AND ROBERTSON, P. Volume rendering on the MasPar MP-1. In In *1992 Workshop on Volume Visualization* (Boston, MA, Oct. 1992), pp. 3–8.

[27] WANG, S., AND KAUFMAN, A. Volume sampled voxelization of geometric primitives. In In *Proceedings of Visualization '93* (San José, CA, Oct. 1993), pp. 78–84.

[28] YAGEL, R., AND KAUFMAN, A. Template-based volume viewing. *Computer Graphics Forum, Proceedings Eurographics 11*, 3 (Sept. 1992), 153–167.

[29] YOO, T. S., NEUMANN, U., FUCHS, H., PIZER, S. M., CULLIP, T., RHOADES, J., AND WHITAKER, R. Direct visualization of volume data. *IEEE Computer Graphics & Appl. 12*, 4 (July 1992), 63–71.